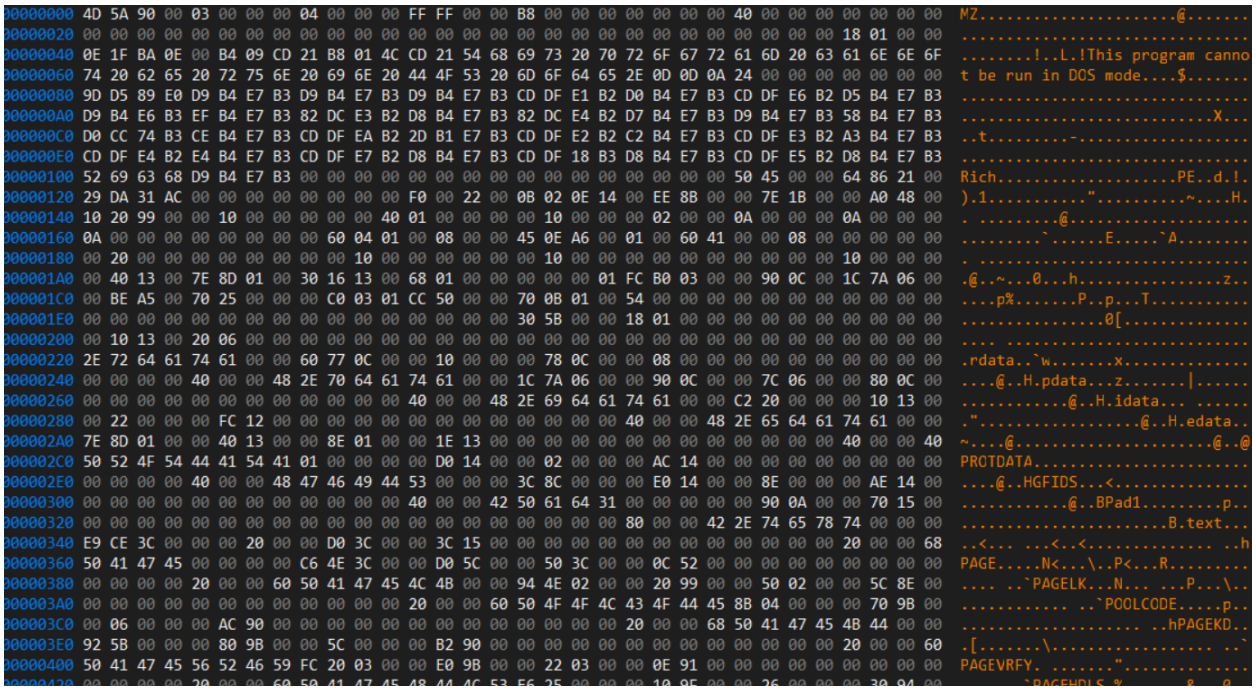
The Portable Executable Journey

Version 1.0
May-2024

By Dr. Shlomi Boutnaru



Screenshot taken from TotalPE

Table of Contents

Table of Contents 2
Introduction 3
DOS Header (struct _IMAGE_DOS_HEADER) 4
DOS Stub 5
NT Headers (struct _IMAGE_NT_HEADERS32/64) 6
File Header (struct _IMAGE_FILE_HEADER) 7
Optional Header (struct _IMAGE_OPTIONAL_HEADER32/64) 8

Introduction

Understanding the PE (Portable Executable) file format is crucial for different tasks such as analyzing malware and reverse engineering. Also, this file format is used by Windows applications, UEFI firmware and even for Linux applications (such as does based on .NET core).

Thus, I wanted to create something that will improve the overall knowledge of the PE file format with writeups that can be read in 1-3 mins. I hope you are going to enjoy the ride.

Lastly, you can follow me on twitter - @boutnaru (https://twitter.com/boutnaru). Also, you can read my other writeups on medium - https://medium.com/@boutnaru. Lastly, You can find my free eBooks at https://TheLearningJourneyEbooks.com.

Lets GO!!!!!!

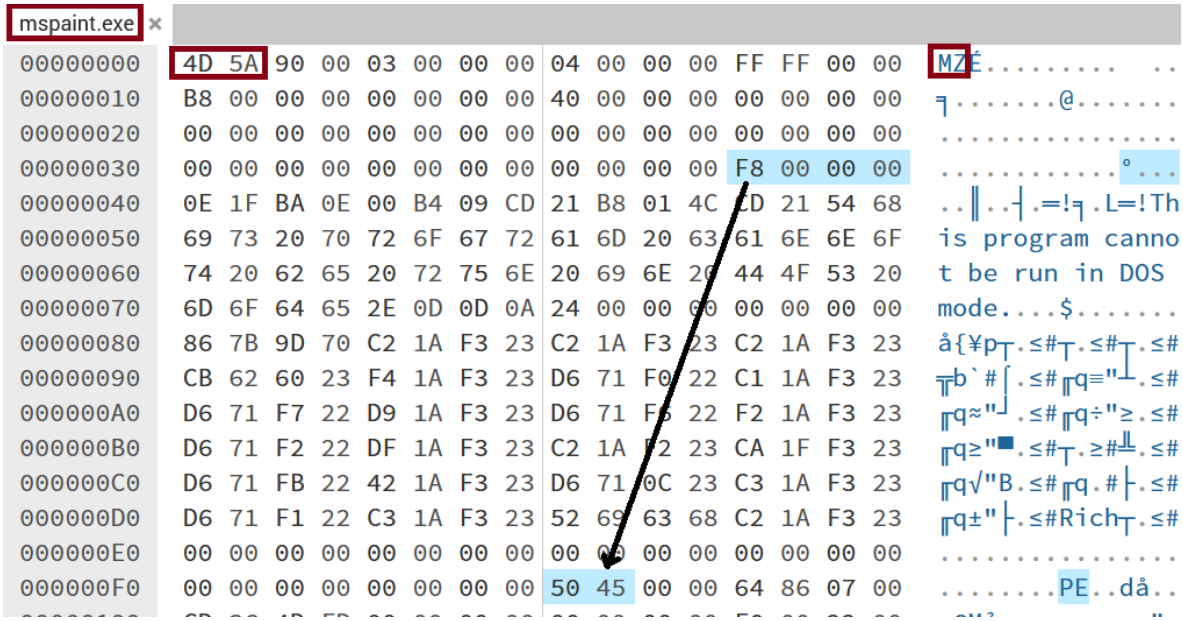
DOS Header (struct _IMAGE_DOS_HEADER)

Every PE (Portable Executable) binary starts with a MS-DOS (Microsoft Disk Operating System) header. By the way, the PE format is used by: Windows 95 and higher, Windows NT 3.1 and higher, ReactOS and UEFI. It is also used by .NET assemblies[1].

Overall, the first two bytes (aka “Magic”) is “0x5A4D” which is “MZ” in ASCII. The letter “MZ” stands for “Mark Zbikowski” , who is one of the designers on the MS-DOS executable - as shown in the screenshot below, taken using “https://hexed.it/” while examining “mspaint.exe”[2].

Moreover, the “DOS Header” is defined using “struct _IMAGE_DOS_HEADER”[3]. As we can see the last field “e_lfanew” contains the file address pointing to the beginning of the PE header (in little endian” - as shown in the screenshot below.

Lastly, the Windows loader cares about “e_magic” and “e_lfanew” from “struct _IMAGE_DOS_HEADER”. The other members (like initial instruction pointer, initial stack pointer, number of pages in the file and more) are relevant for MS-DOS when executing the stud program, which follows the DOS header and is going to be detailed in a future writeup[4].



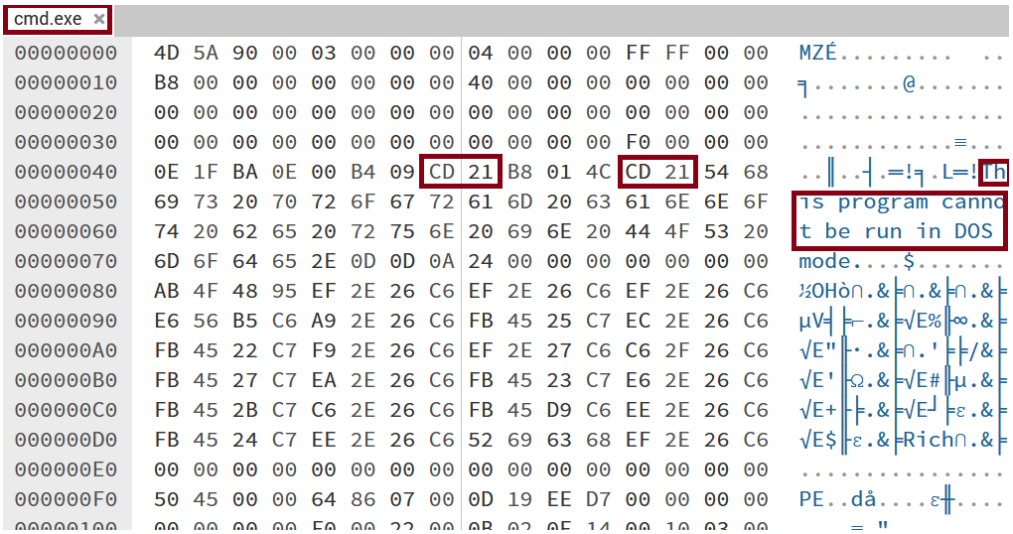
[1] https://wiki.osdev.org/PE
[2] https://medium.com/@boutnaru/the-windows-process-journey-mspaint-exe-paint-3317a8fb3a57
[3] https://github.com/reactos/reactos/blob/master/drivers/filesystems/udfs/Include/ntddk_ex.h#L99
[4] https://osandamalith.com/2020/07/19/exploring-the-ms-dos-stub/

DOS Stub

Just after the "DOS Header"[5] and before the "NT Headers" we have the "DOS Stub". The DOS stub is a program which is invoked in case the file is executed in MS-DOS. By default it displays the following message: "This program cannot be run in MS-DOS mode." - as shown in the screenshot below, taken using "https://hexed.it/" while examining "cmd.exe"[6]. Any valid MS-DOS application can act as the DOS stub program, due to that we can also change it if we want[7].

Moreover, the DOS stub is a 16-bit program (real-mode). Also, in case the stub program is executed the instruction pointer (EIP), stack pointer (ESP), the code segment (CS) and stack segment are initialized based on data stored as part of the DOS header: "e_ip", "e_sp", "e_cs" and "e_ss" respectively[8].

Lastly, we can see in the screenshot below that the opcode "int 0x21" is in use (CD 21). Most of the general functions and services offered by DOS are implemented through this interrupt[9]. In the case of the DOS stub it is used for printing and quitting the program (with an exit code).



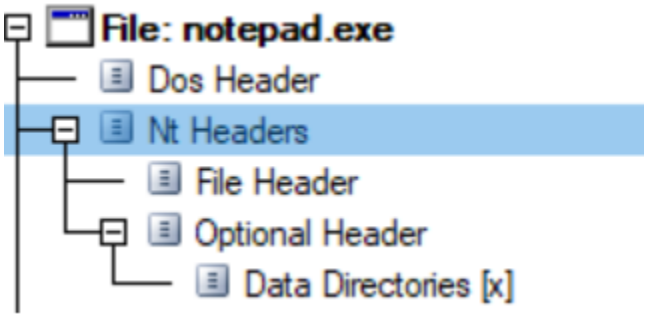
[5] https://medium.com/@boutnaru/the-portable-executable-journey-dos-header-ea5b29f15612
[6] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81
[7] https://learn.microsoft.com/en-us/cpp/build/reference/stub-ms-dos-stub-file-name?view=msvc-170
[8] https://osandamalith.com/2020/07/19/exploring-the-ms-dos-stub/
[9] http://bbc.nvg.org/doc/Master%20512%20Technical%20Guide/m512techb_int21.htm

NT Headers (struct _IMAGE_NT_HEADERS32/64)

Pointed from the “DOS Header”[10] and should be after the “DOS stub”[11] we have the “NT Headers”. The “NT Header” (aka “PE Header”) is defined in one of two data structures: “struct _IMAGE_NT_HEADERS32”[12] for 32-bit binaries and “struct _IMAGE_NT_HEADERS64”[13] for 64-bit binaries.

Overall, the data structure is composed of 3 fields: “Signature”, “File Header” (struct _IMAGE_FILE_HEADER) and “Optional Header“ (struct _IMAGE_OPTIONAL_HEADER32/64). Signature is “PE\0\0”, which are the letters "P" and "E" followed by two null bytes. File header is a standard COFF header - which we are going to detail in a future writeup. The optional header is a must in the case of an image file and optional only in case of an object file[14].

Lastly, because there are two versions for the “Optional Header” (32/64 bit) we also have two versions of the “NT Headers” data structure. We can see the hierarchy of the “NT Headers” in the output of “CFF Explorer” created by Erik Pistelli which allows viewing/modifying PE files[15] - as shown in the screenshot below.



[10] https://medium.com/@boutnaru/the-portable-executable-journey-dos-header-ea5b29f15612
[11] https://medium.com/@boutnaru/the-portable-executable-journey-dos-stub-0ca8cda20570
[12] https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-image_nt_headers32
[13] https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-image_nt_headers64
[14] https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#signature-image-only
[15] https://ntcore.com/explorer-suite/

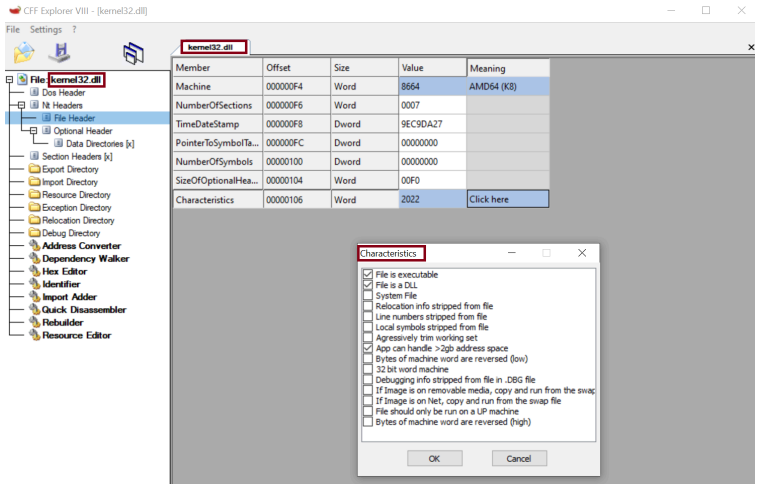
File Header (struct _IMAGE_FILE_HEADER)

“File Header” represents the COFF (Common Object File Format) header. COFF is used for storing compiled code (output of a compiler and linker). Thus, PE (sometimes called PE/COFF) contains a version of COFF. Its goal is to hold basic information about the file while containing pointers to other data structures. By the way, this header is fixed in size[16].

Overall, the “File Header” consists of seven fields: “Machine”, “NumberOfSections”, “TimeDateStamp”, “PointerToSymbolTable”, “NumberOfSymbols”, “SizeOfOptionalHeader” and “Characteristics”. The “Machine” field describes the architecture which the file can be executed on, examples of relevant values are: x86 (IMAGE_FILE_MACHINE_I386 which equals to 0x014c), “Intel Itanium” (IMAGE_FILE_MACHINE_IA64 which equals to 0x0200) and “x64” (IMAGE_FILE_MACHINE_AMD64 which equals to 0x8664) - more values are available in the Microsoft documentation[17]. “NumberOfSections” defines how long is the section table (by the way the limit of the Windows loader is 96). “TimeDateStamp” is the number of seconds (since January 1, 1970 midnight) according to the system clock inserted by the linker[18].

Moreover, “PointerToSymbolTable” is the offset in bytes to the symbol (name to address of variable/function) table (0 if there is no symbol table). “NumberOfSymbols” is the number of symbols in the symbol table. “SizeOfOptionalHeader” is the size of the “Optional Header” in bytes. By the way, if the size is “0” the file is an “object file”[19].

Lastly, the “Characteristics” field is used to describe the attributes of the file like: support of >2GB addresses, debugging information is removed, the file is a DLL and more[20]. An example of parsing the “File Header” for the “kernel32.dll” file using “CFF Explorer” is shown below.

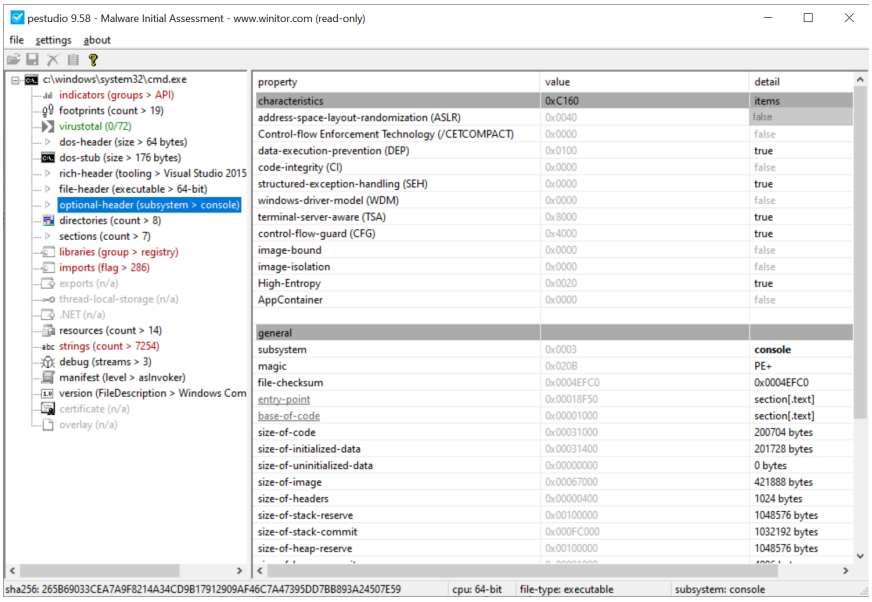


[16] https://wiki.osdev.org/COFF
[17] https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#machine-types
[18] https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-image_file_header
[19] https://gist.github.com/TheWover/730275aedcb4a2413cdbc8a2e33a8df4
[20] https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#characteristics

Optional Header (struct _IMAGE_OPTIONAL_HEADER32/64)

Despite its name the "Optional Header" is not optional (it is required) in case of a compiled and linked binary. This header contains information used by the OS loader when loading a PE file. Because we can have a 64-bit or 32-bit PE file, there are corresponding versions of the "Optional Header" data structure: "struct _IMAGE_OPTIONAL_HEADER64" and "struct _IMAGE_OPTIONAL_HEADER32"[21].

Moreover, remember that the size of the "Optional Header" is not fixed and it is defined in the field "SizeOfOptionalHeader" as part of the "File Header"[22]. They are multiple fields as part of the "Optional Header" - as shown in the screenshot below (taken using pestudio). In general we can divide the optional header to three main parts: standard fields, Windows specific fields and data directories[23].



Thus, the standard fields part includes eight fields. "Magic", which can be "0x10b" (PE) and "0x20b" (PE+), the second allows access to 64-bit address space. "MajorLinkerVersion" and "MinorLinkerVersion" which are the major and minor versions of the linker. "SizeOfCode" which is the total size of all code sections (.text and more if relevant). "SizeOfInitializedData" and "SizeOfUninitializedData" which is the total size of initialized and uninitialized (BSS) sections. "AddressOfEntryPoint" and "BaseOfCode"[24].

[21] https://github.com/reactos/reactos/blob/master/sdk/include/host/pecoff.h#L139
[22] https://medium.com/@boutnaru/the-portable-executable-journey-file-header-struct-image-file-header-00360271f147
[23] https://learn.microsoft.com/en-us/windows/win32/debug/pe-format
[24] https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

Lastly, the Windows specific field part contains 21 fields. “ImageBase”, “SectionAlignment, “FileAlignment”, “MajorOperatingSystemVersion”, “MinorOperatingSystemVersion “, “MajorImageVersion “, “MinorImageVersion”, “MajorSubsystemVersion“, “MinorSubsystemVersion”, “Win32VersionValue” (reserved, must be “0”), “SizeOfImage”, “SizeOfHeaders “, “CheckSum”, “Subsystem”, “DllCharacteristics”, “SizeOfStackReserve”, “SizeOfStackCommit”, “SizeOfHeapReserve”, “SizeOfHeapCommit”, “LoaderFlags” (reserved, must be zero) and “NumberOfRvaAndSizes”[25].

[25] https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-image_optional_header64